**International University**

School of Computer Science and Engineering

**Object Oriented Programming**

**IT069IU**

**Project: Pokemon RPG Game**

**Submitted by**

Đỗ Hùng Việt – ITCSIU22197

Phan Đức Trí – ITCSIU22151

Nguyễn Anh Trí – ITCSIU22152

Hà Minh Trí – ITCSIU22194

**Date Performed:** 15/01/2024

**Date Submitted:** 15/01/2024

**Course Instructor:** Mr. Tran Thanh Tung

**TABLE OF CONTENTS**

[**LIST OF FIGURES** 3](#_Toc156254896)

[**I.** **Introduction** 5](#_Toc156254897)

[**1.** **Objectives:** 5](#_Toc156254898)

[**2.** **Tools used:** 5](#_Toc156254899)

[**II. Property of Game: 6**](#_Toc156254900)

[**1. Goal: 6**](#_Toc156254901)

[**2. Rule 6**](#_Toc156254902)

[**III. UML Diagram: 7**](#_Toc156254903)

[**IV. Main game tree: 7**](#_Toc156254904)

[**V. Methodology 13**](#_Toc156254905)

[**1. Gameplay 13**](#_Toc156254906)

[**2. Design 14**](#_Toc156254907)

[**a.** **Main character:** 15](#_Toc156254908)

[**b.** **Enemies:** 16](#_Toc156254909)

[**c.** **Objects:** 17](#_Toc156254910)

[**d.** **Map:** 18](#_Toc156254911)

[**e.** **User Interface:** 19](#_Toc156254912)

[**f.** **Audio:** 22](#_Toc156254913)

[**VI. Demo: 22**](#_Toc156254914)

[**VII. Conclusion: 22**](#_Toc156254915)

[**1. Result: 22**](#_Toc156254916)

[**2. Limitation: 22**](#_Toc156254917)

[**VIII. References: 22**](#_Toc156254918)

# **LIST OF FIGURES**

Figure 1. Charmander 5

Figure 2. The demonstration of Pokemon RPG game 6

Figure 3. The UML Diagram 7

Figure 4. First sight of the game in the demo map 14

Figure 5. The health and energy potion 14

Figure 6. The barrels used for the game 14

Figure 7. The Charmander images taken from Pokemon Mystery Dungeon 15

Figure 8. Charmander’s sprite 16

Figure 9. Smeargle’s sprite 16

Figure 10. Skuntank’s sprite 16

Figure 11. Mewtwo’s sprite 17

Figure 12. Victreebel’s sprite 17

Figure 13. Bullet used by Victreebel 17

Figure 14. Trap 17

Figure 15. Energy (above) and health (below) potions 17

Figure 16. Barrels for energy potion (above) and health potion (below) 17

Figure 17. Tutorial map 18

Figure 18. First map 18

Figure 19. Second map 18

Figure 20. Third map 18

Figure 21. Gameplay background 18

Figure 22. Starting background 19

Figure 23. Small cloud 19

Figure 24. Tilesset 19

Figure 25. Button Quit 19

Figure 26. Button Voulme 20

Figure 27. Button Sound 20

Figure 28. Button URM 20

Figure 29. Game completed 20

Figure 30. Game lost 20

Figure 31. Menu background 21

Figure 32. Option menu 21

Figure 33. Pause menu 21

Figure 34. Health and energy bar 21

Figure 35. Audio list 22

# **Introduction**

* For the project of Object – Oriented Programming class, we have conducted a project which is using basic and kind of advantaged of OOP knowledge.

1. **Objectives:**

* Pokemon is a famous Japanese media franchise consisting of famous video games, animated series and films, etc… One of the most famous Pokemon is Charizard - a lizard Pokemon. We use its baby model Charmander as the main character for this game. Our team is all impressed by the gameplay of Mario. So we made a game influenced by Mario but added more Pokemons as enemies to make the difference.

Figure 1. Charmander

* The project aims to recreate a fully playable game based on the original Mario game but modified as an RPG game. In short, this project aims to:
* Create a redesigned game to entertain.
* Practicing OOP techniques.
* Have a brief look at game development, code optimization, and project management.
* Evaluate the ability to build more features on the base program.

1. **Tools used:**

* Programming languages used: Java.
* IDEs for programming and debuging: eclipse.
* Project management: Google Drive, GitHub.
* Communication: Discord, Messenger.
* Map Generator: GNU Image Manipulation Program (GIMP).

# **Property of Game:**

1. **Goal:**

* The aim of this game is just like Mario. You need to pass through the end of the map. However, we decided to make this game a little challenging so we added enemies that can hit the player. We designed a total of 2 levels.

1. **Rule**

* Preventing yourself from falling off the edge and keeping the health point slider above zero.
* Destroyed boxes can drop items to help you recover your health and energy.
* Deafeting all the enemies.

Figure 2. The demonstration of Pokemon RPG game

# **UML Diagram:**

Figure 3. The UML Diagram

# **Main game tree:**

* Main:
* class Game Type():
* Acts as the central class in charge of the game's overall flow.
* Creates and manages 'Menu', 'Playing', 'GameOptions', and 'SoundOptions' instances.
* Implements the game loop, which handles updates and rendering for various game states.
* Manages user inputs via 'KeyBoardInputs' and 'MouseInputs' instances.
* Accesses various game components such as the menu, playing state, and sound player.
* class GamePanel Type():
* Extends 'JPanel' to represent the game's main panel.
* Manages user inputs via 'KeyBoardInputs' and 'MouseInputs' instances.
* Uses the 'Graphics' object to render the game using the 'paintComponent' method.
* Allows access to the 'Game' instance for rendering purposes.
* class GameWindow Type():
* Creates and maintains the game window (JFrame).
* Configures the window's properties such as size, location, and visibility.
* Adds a 'WindowFocusListener' to handle focus events and delegate them to the game for further processing.
* MainClass(): Contains the'main' method, which serves as the application's entry point.
* Initializes and starts the game by creating an instance of the 'Game' class.
* Gamestate:
* class Gameoptions(): Manages game settings and options.
* Takes user input for sound settings and menu navigation.
* class GameState Enum():Counts the number of game states (PLAYING, MENU, OPTIONS, QUIT).
* class Menu():
* Represents the game's main menu.
* Handles user input for menu navigation and game state transitions.
* class Participation():
* Represents the game state while the player is actively playing.
* Controls the player, the level, the enemies, the objects, and the overlays.
* Takes user input for movement, attacks, and interactions.
* class State():
* Serves as a base class for various game states.
* Provides common functionality for switching between game states.
* class StateMethods Interface(): Specifies the methods that must be implemented by classes representing various game states.
* Entity:
* class EnemyManager():
* Manages and updates the game's various enemies (Painters, Skunks, Bosses).
* Loads enemy images for various levels.
* Makes enemies appear on the screen.
* Looks for enemy hits caused by the player's attacks.
* class Entity():
* An abstract class that is the foundation for all game entities (including the player and enemies).
* Manages entity position, size, hitboxes, and animation states.
* class Painter():
* Extends the 'Enemy' class and represents a specific type of enemy in the game ("Painter").
* Adds behavior and updates to the Painter enemy, such as movement and attacks.
* class Player():
* Extends the 'Entity' class and represents the game's player character.
* Oversees player-specific features such as health, energy, animations, and controls.
* Handles input from the player and updates the player's state, position, and animation.
* class Enemy():
* Extends the 'Entity' class and serves as the foundation for various enemy types (Painter, Skunk, and Boss).
* Provides enemies with common functionality such as behavior updates, hit detection, and attack handling.
* class Boss():
* Extends the 'Enemy' class by representing a specific type of enemy ("Boss") in the game.
* Implements Boss enemy behavior and updates, such as movement and attacks.
* class Skun():
* Extends the 'Enemy' class and represents a particular type of enemy ("Skunk") in the game.
* Adds Skunk enemy behavior and updates, such as movement and attacks.
* Audio:
* The SoundPlayer class handles background music and sound effects in the game. It includes features for loading, playing, pausing, and adjusting the volume of sounds. It includes background music for each game level, random attack sounds, and level completion sounds. It also allows you to mute songs and effects independently.
* Inputs:
* To handle keyboard input events, the 'KeyBoardInputs' class implements the 'KeyListener' interface. It routes input events to specific methods based on the current game state ('MENU', 'PLAYING', or 'OPTIONS'). The methods are in charge of processing key presses and releases in the appropriate game states.
* To handle mouse input events, the 'MouseInputs' class implements the 'MouseListener' and 'MouseMotionListener' interfaces. It directs mouse events to specific methods based on the current game state, similar to 'KeyBoardInputs'. The methods handle mouse movements, clicks, presses, and releases in various game states ('MENU', 'PLAYING', or 'OPTIONS').
* Level:
* class Level():
* Represents a game level and is in charge of loading and managing the level's various elements.
* Reads from an image level data, entities (such as enemies and the player), and objects (such as potions and traps).
* Calculates level offsets and provides methods for retrieving level information such as sprite indices, level data, and entity spawn points.
* class LevelManager():
* Manages the game's multiple levels and handles the transition between them.
* Loads level sprites and creates a collection of Level instances.
* Provides methods for loading the next level, drawing the current level, and retrieving current level information.
* Manages and organizes the game's levels by using Level class instances.
* UI:
* class GameEndOverlay():
* Represents the game overlay that appears when the game is over.
* Has buttons for going back to the menu and replaying the game.
* class LvlComOverlay():
* This is the overlay that appears when a level is completed.
* Has buttons for progressing to the next level and returning to the menu.
* class MenuButton():
* Represents a game menu button.
* Handles menu button appearance and behavior, including state changes.
* class PauseButton():
* Serves as the base class for pause menu buttons.
* Contains properties and methods that are shared by pause menu buttons.
* class PauseOverlay():
* This class represents the pause menu overlay that appears during the game.
* Has buttons for going back to the menu, replaying the game, and unpausing.
* class SoundButton():
* Represents a button in the pause menu that toggles sound (music or sound effects).
* Controls how sound buttons appear and behave.
* class SoundOptions():
* Controls sound-related options in the pause menu, such as volume.
* Has buttons for switching between music and sound effects, as well as a volume control slider.
* class UrmButton():
* Represents a button in the game overlay with Urm images (possibly characters or icons).
* Controls how Urm buttons appear and behave.
* class VolumeButton():
* Represents a volume control button in the pause menu with a slider.
* Provides a slider for users to adjust the volume level and handle related interactions.
* Object:
* class GameContainer():
* This class represents game containers (such as boxes and barrels).
* Manages container hitbox creation and updates.
* class GameObject():
* An abstract class that represents game objects that share attributes such as position, hitbox, and animation.
* Handles animation, hitbox initialization, and game object updates.
* class ObjectManager():
* Controls various in-game objects like potions, containers, and projectiles.
* Handles object collision detection and updates.
* class Potion():
* In the game, this represents potions.
* Derives from GameObject and adds potion-specific functionality such as hovering.
* class Projectile():
* Denotes projectiles fired by shooters.
* Controls projectile movement and state.
* class Shooter():
* Represents in-game shooter objects.
* Oversees shooter hitbox initialization and updates.
* class StepTrap():
* Represents the game's step traps.
* In charge of step trap hitbox initialization.
* Util:
* class Constants(): Stores world rules such as gravity, enemy damage, sprite sizes, and more. Consider a rulebook for the game's physics and graphics.
* class HelpMethods(): Serves as the game's toolbox. It determines whether objects can move, whether bullets hit walls, and whether enemies can see the player. It's the driving force behind many game mechanics.
* Class LoadSave(): Downloads all of the images and levels required by the game, such as character sprites, backgrounds, and enemy images. It's similar to a library where the game stores its visual assets.

# **Methodology**

1. **Gameplay**

* The gameplay is simple: You as Charmander are spawned on the map, and the player can only move left-right and jump. In this game, you have to defeat all the enemies. Only Victreebel cannot be hit so you just need to avoid all of its attacks. It is also the one that has a long-range attack. You can hit the enemies consecutively by using strike and while the enemies are under your attack, they cannot move or hit you. Another type of attack you can use is quick strike but it will consume a little energy. The player must clear all the monsters except Victreebel to move on to the next level. The player can destroy the boxes along the path to get useful items:
* Red potion: Restore health.
* Blue potion: Restore energy.
* After finishing the first level, the player has to face an extremely long map. This map is not only longer but also has a new enemy to make the game more difficult.
* A video game with a mountain and trees

  Description automatically generatedThe player uses ‘A’ and “D’ keys to move left and right, respectively. ‘Spacebar’ is used to jump. Clicking the mouse to attack the enemies.

Figure 4. The barrels used for the game

Figure 5. The health and energy potion

Figure 6. First sight of the game in the demo map

1. **Design**

* Our first idea for this game is to make its gameplay the same as Pokemon Mystery Dungeon. So, we use the sprites of the said game for the main character and its enemies The sprites we found on this web are very useful because it has a lot of sprites in any direction so we do not have to spend so much time to do so.
* We tried to design some sprites ourselves but we could not have the models satisfy us. Therefore, we found some online sources so we could have the design material match with our game and we changed a little bit to make the difference.

1. **Main character:**

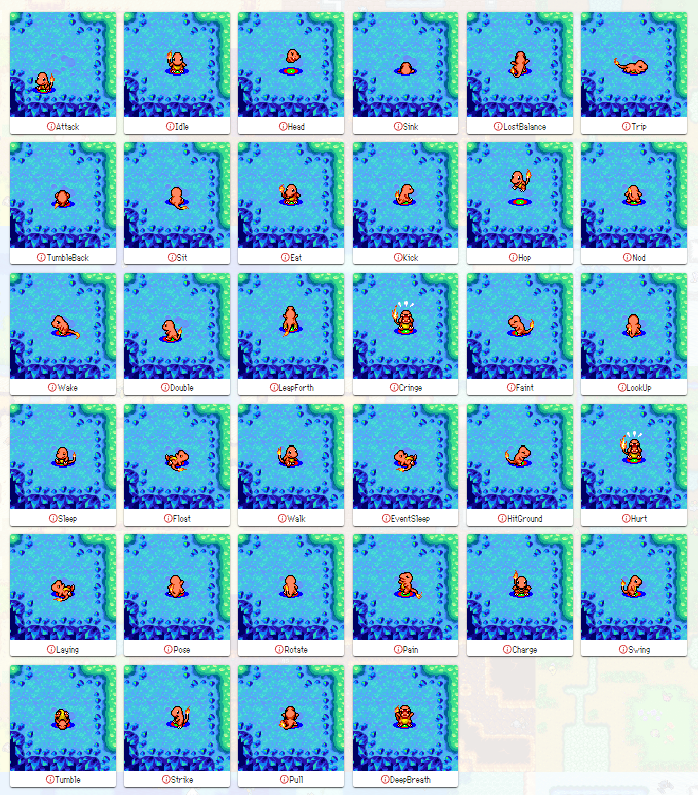
* Charmander sprite taken from Pokemon Mystery Dungeon:

Figure 7. The Charmander images taken from Pokemon Mystery Dungeon

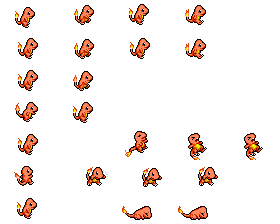
* Charmander sprtie using for the project:

Figure 8. Charmander’s sprite

1. **Enemies:**

* A screenshot of a video game

  Description automatically generatedSmeargle:

Figure 9. Smeargle’s sprite

* Skuntank:

Figure 10. Skuntank’s sprite

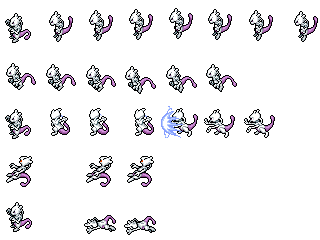
* Mewtwo:

Figure 11. Mewtwo’s sprite

* Victreebel:

Figure 12. Victreebel’s sprite

1. **Objects:**

* Bullet:

Figure 13. Bullet used by Victreebel

* Traps:

Figure 14. Trap

* Potions:

Figure 15. Energy (above) and health (below) potions

* Barrels:

Figure 16. Barrels for energy potion (above) and health potion (below)

1. **Map:**

* Tutorial map, 53 pixels in width and 14 pixels in height:

Figure 17. Tutorial map

* First map, 50 pixels in width and 14 pixels in height:

Figure 18. First map

* Second map, 200 pixels in width and 14 pixels in height:

Figure 19. Second map

* Third map, 200 pixels in width and 14 pixels in height:

Figure 20. Third map

* Gameplay background:

Figure 21. Gameplay background

* Starting background:



Figure 22. Starting background

* Small cloud:

Figure 23. Small cloud

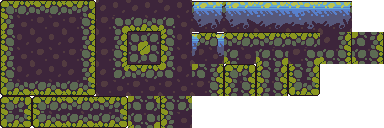
* Tileset:

Figure 24. Tilesset

1. **User Interface:**

* Button Quit:

Figure 25. Button Quit

* Button Sound:
* Button Volume:

Figure 26. Button Voulme

Figure 27. Button Sound

* Button URM:.

Figure 28. Button URM

* Game completed:

Figure 29. Game completed

* A wooden sign with text

  Description automatically generatedGame lost:

Figure 30. Game lost

* Menu background:

Figure 31. Menu background

* Option menu:

Figure 32. Option menu

* Pause menu:

Figure 33. Pause menu

* Health and energy bar:

Figure 34. Health and energy bar

1. **A screenshot of a computer

   Description automatically generatedAudio:**

Figure 35. Audio list

# **Demo:**

* Video demo uploaded on Google Drive: <https://drive.google.com/file/d/1-X9uYgF4xQ0ctH0tfCarvkf2gi12YxRP/view?usp=sharing>

# **Conclusion:**

1. **Result:**

* Our group project has been built with basic rules based on the principle of the OOP method. The combination of classes works effectively. Due to limited knowledge and time, our game had some problems that we could not fix on time.

1. **Limitation:**

* The movement of the character is not as smooth as we expected.
* We did not find the perfect sound for our game.
* We did not work well on GitHub.
* We should include the shooting mode to make the game more interesting.

# **References:**

Kaarin (Jul 4, 2023), Platformer Tutorial. - Java Retrieved from: <https://www.youtube.com/playlist?list=PL4rzdwizLaxYmltJQRjq18a9gsSyEQQ-0>

Sprites used in the project. Retrieved from: <https://github.com/PMDCollab> and <https://sprites.pmdcollab.org/>

Map inspired from AdvancedFan2020 (Jul 29, 2023), Mega Man Maker. Retrieved from: <https://www.deviantart.com/advancedfan2020/art/Mega-Man-Maker-Map-542399-969511117>

Map inspired from Revned (2008), Mega Man. Retrieved from: <https://revned77.github.io/games/MegaMan.html>